PURDUE UNIVERSITY

# Applied Quantum Computing III

## ECE 50652

## Spring 2026

Recitation Module 1: Qiskit

**Instructor: Dr. Shengwang Du**

Recitation: Monday 2:30pm-3:20pm

Knoy Hall of Technology B033

# Update on Qiskit code from Lecture Notes

In Qiskit 1.0 and later versions, the execute function has been removed. The recommended approach is to use the transpile function followed by the backend's run() method.

```
# Legacy path (deprecated)
from qiskit import execute
job = execute(circ, backend)

# New, recommended path
from qiskit import transpile
new_circuit = transpile(circ, backend)
job = backend.run(new_circuit)
```

# Update on Qiskit code from Lecture Notes

The + operator is no longer supported by QuantumCircuit

```
# Legacy path (deprecated)
qc = circ + meas

# New, recommended path
qc = circ.compose(meas)
```

# Example

- Hello.ipynb
- Qiskit_Starter_01_new.ipynb

Qiskit installation

**Steps for Installation**

- **Prerequisites:** Python 3.9 or higher is required.
- **Create Environment (Recommended):**

  bash

  ```
  python3 -m venv qiskit-env
  source qiskit-env/bin/activate  # On Windows: qiskit-env\Scripts\activate
  ```

- **Install Qiskit:**

  bash

  ```
  pip install qiskit
  ```

- **Install Visualization Support:**

  bash

  ```
  pip install 'qiskit[visualization]'
  ```

**Optional Packages**

- **Qiskit Aer (Simulators):** `pip install qiskit-aer`.
- **Qiskit Machine Learning:** `pip install qiskit-machine-learning`.
- **Qiskit Algorithms:** `pip install qiskit-algorithms`.

To verify the installation, you can run `pip show qiskit` in your terminal.

```
pip install matplotlib
```

1. **For IBM Quantum hardware:** `pip install qiskit-ibm-runtime`
2. **Install/Open Jupyter Notebook:**
3. bash
4. ```
   pip install jupyter
   ```
5. ```
   jupyter notebook
   ```
6.

Run Qiskit code in your computer:

```
python3 -m venv qiskit-env
source qiskit-env/bin/activate  # On Windows: qiskit-env\Scripts\activate
jupyter notebook
```

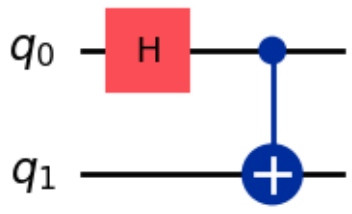
For the first time
#You need to run this QiskitRuntimeService.save_account for the first time in your local computer to save your account information.
#After the first time, you do not need to run it again.
#QiskitRuntimeService.save_account(
#    channel="ibm_quantum_platform", # optional
#    token="<??????>", # Replace with your actual API key
#    instance="CRN"        # Optional: Replace with your instance CRN if you have one
#)

# Hello

March 29, 2026

[1]:
```
from qiskit import QuantumCircuit
from qiskit.quantum_info import SparsePauliOp
from qiskit.transpiler import generate_preset_pass_manager
from qiskit_ibm_runtime import QiskitRuntimeService
from qiskit_ibm_runtime import EstimatorOptions
from qiskit_ibm_runtime import EstimatorV2 as Estimator
from matplotlib import pyplot as plt
# Uncomment the next line if you want to use a simulator:
# from qiskit_ibm_runtime.fake_provider import FakeBelemV2


# Create a new circuit with two qubits
qc = QuantumCircuit(2)

# Add a Hadamard gate to qubit 0
qc.h(0)

# Perform a controlled-X gate on qubit 1, controlled by qubit 0
qc.cx(0, 1)

# Return a drawing of the circuit using MatPlotLib ("mpl").
# These guides are written by using Jupyter notebooks, which
# display the output of the last line of each cell.
# If you're running this in a script, use `print(qc.draw())` to
# print a text drawing.
qc.draw("mpl")
```

[1]:

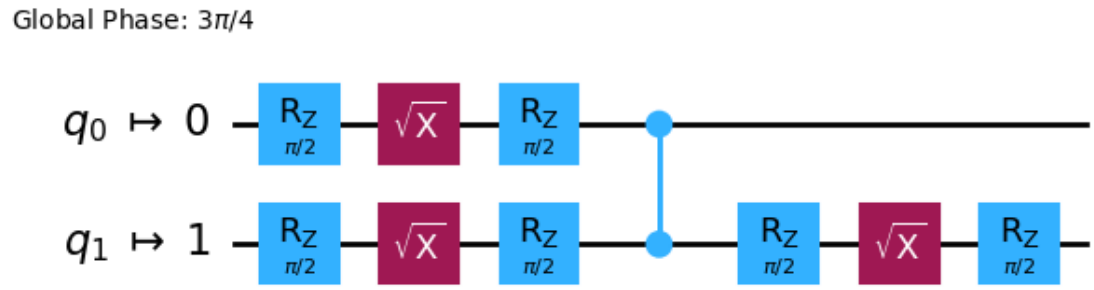


[2]:
```
# Set up six different observables.

observables_labels = ["IZ", "IX", "ZI", "XI", "ZZ", "XX"]
observables = [SparsePauliOp(label) for label in observables_labels]
```

[4]:
```
#You need to run this QiskitRuntimeService.save_account for the first time in your local computer to save your account information.
#After the first time, you do not neet to run it again.
#QiskitRuntimeService.save_account(
#    channel="ibm_quantum_platform", # optional
#    token="WHooKhCnTBjiUncBubFk3FshGGlxguA6XRNAUm29ytvv", # Replace with your actual API key
#    instance="CRN"         # Optional: Replace with your instance CRN if you have one
#)

service = QiskitRuntimeService()

backend = service.least_busy(simulator=False, operational=True)

# Convert to an ISA circuit and layout-mapped observables.
pm = generate_preset_pass_manager(backend=backend, optimization_level=1)
isa_circuit = pm.run(qc)

isa_circuit.draw("mpl", idle_wires=False)
```

```
qiskit_runtime_service.__init__:WARNING:2026-03-29 15:04:23,156: Instance was
not set at service instantiation. Free and trial plan instances will be
prioritized. Based on the following filters: (tags: None, region: us-east, eu-
de), and available plans: (open), the available account instances are: Purdue-
AQCIII. If you need a specific instance set it explicitly either by using a
saved account with a saved default instance or passing it in directly to
QiskitRuntimeService().
qiskit_runtime_service.backends:WARNING:2026-03-29 15:04:23,981: Loading
instance: Purdue-AQCIII, plan: open
qiskit_runtime_service.backends:WARNING:2026-03-29 15:04:27,455: Using instance:
Purdue-AQCIII, plan: open
```

[4]:



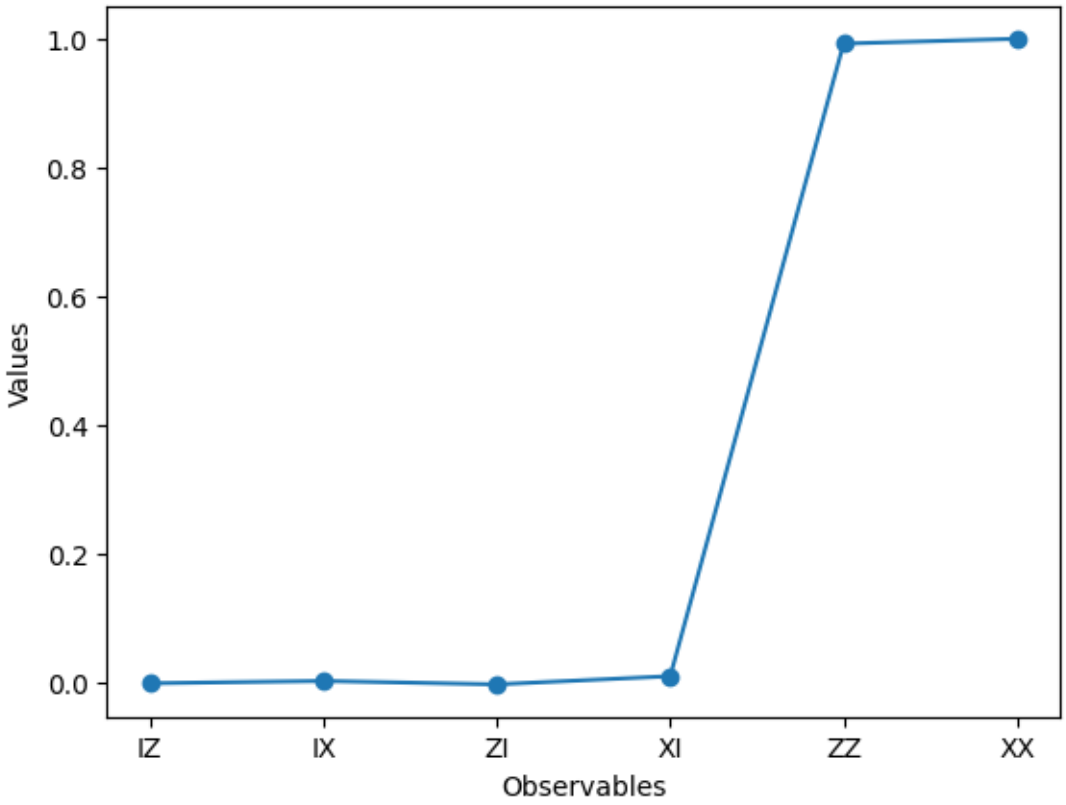

[5]:
```
# Construct the Estimator instance.

estimator = Estimator(mode=backend)
estimator.options.resilience_level = 1
estimator.options.default_shots = 5000

mapped_observables = [
    observable.apply_layout(isa_circuit.layout) for observable in observables
]

# One pub, with one circuit to run against five different observables.
job = estimator.run([(isa_circuit, mapped_observables)])

# Use the job ID to retrieve your job data later
print(f">>> Job ID: {job.job_id()}")
```

```
>>> Job ID: d74ng2i3qcgc73fqq3jg
```

[6]:
```
# This is the result of the entire submission.  You submitted one Pub,
# so this contains one inner result (and some metadata of its own).
job_result = job.result()

# This is the result from our single pub, which had six observables,
# so contains information on all six.
pub_result = job.result()[0]
```

[7]:
```
# Plot the result

values = pub_result.data.evs

errors = pub_result.data.stds

# plotting graph
plt.plot(observables_labels, values, "-o")
plt.xlabel("Observables")
```

```
plt.ylabel("Values")
plt.show()
```




[ ]:

# Qiskit Migration Overview

This document summarizes the main changes required when migrating older Qiskit notebooks to the current Qiskit and IBM Runtime workflow.

## 1 Overview

1. **Top-level legacy imports were removed or replaced**

   In older code, it was common to write:

```
from qiskit import execute, Aer, IBMQ, assemble
```

   In the current workflow:

   - `Aer` is imported from `qiskit_aer`
   - `IBMQ` is replaced by `QiskitRuntimeService`
   - `execute` is no longer the standard workflow
   - `assemble` is no longer used in the old way

   A typical replacement is:

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit_ibm_runtime import QiskitRuntimeService
```

2. **Simulator access changed**

   Older notebooks often used:

```
backend = Aer.get_backend('qasm_simulator')
```

   or

```
svsim = Aer.get_backend('statevector_simulator')
```

   In current Qiskit, the preferred simulator interface is `AerSimulator()`, for example:

```
from qiskit_aer import AerSimulator

backend = AerSimulator()
```

   For statevector workflows, one typically uses:

```
svsim = AerSimulator(method="statevector")
qc.save_statevector()
result = svsim.run(qc).result()
state = result.get_statevector()
```

3. **The old `execute(...)` and `assemble(...)` pattern was removed**

   Older code often followed:

   ```
   qobj = assemble(circuit, shots=shots)
   result = backend.run(qobj).result()
   ```

   This was replaced by a direct transpile-and-run pattern on simulators:

   ```
   t_circuit = transpile(circuit, backend)
   result = backend.run(t_circuit, shots=shots).result()
   ```

   This was one of the most common updates across the notebooks.

4. **Circuit addition with `+=` was replaced by `compose()`**

   In older notebooks, two circuits were sometimes combined using:

   ```
   qc += oracle
   ```

   In current Qiskit, this is replaced by:

   ```
   qc = qc.compose(oracle)
   ```

   This change appeared in oracle-based algorithms such as Simon and Deutsch–Jozsa.

5. **Identity-gate placeholders were simplified or removed**

   Some older notebooks explicitly inserted identity gates such as:

   ```
   qc.i(qubit)
   ```

   In the updated workflow, these were usually removed entirely when they were logically unnecessary. In many oracle constructions, doing nothing on a branch is better expressed by simply omitting that operation.

6. **IBMQ provider-based access was replaced by Runtime service access**

   Older code often used:

   ```
   IBMQ.load_account()
   provider = IBMQ.get_provider(hub='ibm-q')
   ```

   In current IBM Runtime usage, this becomes:

   ```
   from qiskit_ibm_runtime import QiskitRuntimeService
   service = QiskitRuntimeService()
   ```

   Backend selection is then done through the service object, commonly with:

```
backend = service.least_busy(
    min_num_qubits=n,
    operational=True,
    simulator=False
)
```

7. **Backend selection became more explicit**

   In older code, backend selection often depended on:

   ```
   from qiskit.providers.ibmq import least_busy
   ```

   In the current workflow, one uses either:

   ```
   backend = service.least_busy(...)
   ```

   or explicitly filters available backends:

   ```
   candidate_backends = [
       b for b in service.backends(simulator=False, operational=True)
       if b.num_qubits >= required_qubits
   ]
   ```

   This was especially useful when older code assumed a backend name that is no longer available.

8. **Job monitoring with `job_monitor` was removed**

   Older notebooks often used:

   ```
   from qiskit.tools.monitor import job_monitor
   job_monitor(job)
   ```

   This module is no longer part of the current workflow. Instead, the job identifier is typically printed:

   ```
   print(job.job_id())
   ```

   and status can be checked directly from the job object if needed.

9. **IBM hardware execution moved to Runtime V2 primitives**

   A major architectural change is that hardware jobs are no longer submitted using:

   ```
   job = backend.run(...)
   ```

   for IBM Runtime backends. Instead, the recommended workflow is:

   (a) generate an ISA-compatible circuit with a preset pass manager,

   (b) submit the circuit with `SamplerV2` or `EstimatorV2`.

   A typical pattern is:

```
from qiskit.transpiler import generate_preset_pass_manager
from qiskit_ibm_runtime import SamplerV2 as Sampler

pm = generate_preset_pass_manager(backend=backend, optimization_level=3)
isa_circuit = pm.run(circuit)

sampler = Sampler(mode=backend)
job = sampler.run([isa_circuit], shots=1024)
```

10. **Result extraction format changed for Runtime jobs**

    Older code often assumed:

    ```
    counts = job.result().get_counts()
    ```

    For current Sampler V2 jobs, the counts are typically extracted from the classical register data, for example:

    ```
    results = job.result()
    counts = results[0].data.c.get_counts()
    ```

    This change appeared repeatedly in the hardware-execution sections.

11. **Statevector workflows became more explicit**

    In older notebooks, statevectors were often obtained through legacy backend calls. In the newer workflow:

    - the statevector simulator is chosen explicitly with `method="statevector"`,
    - the circuit explicitly saves the statevector,
    - the result is extracted from the saved data.

    This made statevector simulation clearer and more robust in several notebooks.

12. **Aqua, Ignis, and older application modules were replaced**

    Some chemistry and VQE notebooks relied on:

    ```
    qiskit.aqua
    qiskit.chemistry
    qiskit.ignis
    ```

    These have been replaced by newer packages and namespaces, such as:

    - `qiskit_nature` for chemistry and second-quantization workflows,
    - `qiskit_algorithms` for VQE and optimizers,
    - direct primitive usage instead of `QuantumInstance`.

    For example:

    ```
    from qiskit_nature.second_q.drivers import PySCFDriver
    from qiskit_algorithms import VQE, NumPyMinimumEigensolver
    from qiskit_algorithms.optimizers import SPSA
    ```

This was one of the biggest structural changes in the VQE notebook.

13. **QuantumInstance-based workflows were removed**

    Older VQE-style code often built a `QuantumInstance` with backend, noise model, and measurement mitigation options. In current code, the flow is usually simpler:

    - build a simulator or primitive directly,
    - pass the primitive to the algorithm.

    For example:

```
from qiskit.primitives import StatevectorEstimator
estimator = StatevectorEstimator()
vqe = VQE(estimator, ansatz, optimizer)
```

14. **Noise-model simulation became more direct**

    Instead of configuring older provider-style noise workflows, the updated pattern uses:

```
from qiskit_aer.noise import NoiseModel
noise_model = NoiseModel.from_backend(device)
backend = AerSimulator(noise_model=noise_model,
                       coupling_map=device.coupling_map)
```